Achieving Quality in Open Source Software

Mark Aberdour, *opensourcetesting.org*

Reviewing objective studies of open source software quality can help us better understand how to achieve software quality in both open and closed-source development.

The open source software community has published a substantial body of research on OSS quality. Focusing on this peer-reviewed body of work lets us draw conclusions from empirical data about how to achieve OSS quality, rather than relying on the large volume of evangelical opinion that has historically dominated this field. This body of published research has become much more critical and objective in its efforts to understand OSS development, and a consensus has emerged on the key components of high-quality OSS delivery.

This article reviews this body of research and draws out lessons learned, investigating how the approaches for delivering high-quality OSS differ from, and can be incorporated into, closed-source software development.

Quality models

We can break the traditional, or closed-source, software engineering quality model into two broad areas: *quality assurance* and *quality control*. Quality assurance occurs throughout the software organization and focuses on process and procedure, learning from mistakes, and ensuring good management practice. Quality control is the process of verification and validation, usually within a structured testing process, using high-level plans and detailed test scripts to document and manage the testing process.

OSS development casts aside traditional notions of quality assurance and control, and a review of the body of research shows a wide range of deviations. Table 1 demonstrates this by comparing quality management in open source and closed-source software development.

OSS development must also manage a geographically distributed team, requiring focus on coordination tasks. Yet OSS development seems to eschew best practices without software quality suffering. Indeed, an extensive study of 100 open source applications found that structural code quality was higher than expected and comparable with commercially developed software.[1] The body of research demonstrates that OSS development does retain some of the underlying best-practice tasks from closed-source development—central management, code ownership, task ownership, planning and strategy, system testing, leadership, and decision making—but these tasks are executed differently.

Further analysis of the differences shows that many of OSS development's deviations are procedural—such as risk assessment, measurable goals and milestones, early defect discovery, quality metrics, and planning and scheduling—that help meet that key traditional software de-

0740-7459/07/$20.00 © 2007 IEEE

Table 1

Quality management in open source and closed-source software development

Closed source	Open source
Well-defined development methodology	Development methodology often not defined or documented
Extensive project documentation	Little project documentation
Formal, structured testing and quality assurance methodology	Unstructured and informal testing and quality assurance methodology
Analysts define requirements	Programmers define requirements
Formal risk assessment process—monitored and managed throughout project	No formal risk assessment process
Measurable goals used throughout project	Few measurable goals
Defect discovery from black-box testing as early as possible	Defect discovery from black-box testing late in the process
Empirical evidence regarding quality used routinely to aid decision making	Empirical evidence regarding quality isn't collected
Team members are assigned work	Team members choose work
Formal design phase is carried out and signed off before programming starts	Projects often go straight to programming
Much effort put into project planning and scheduling	Little project planning or scheduling

velopment goal, the release deadline. As concrete release deadlines are less of an issue in OSS development, it might be that these deviations don't negatively impact software quality. Current research excludes this area, but it would make an interesting area for further study.

OSS quality management

OSS development relies on understanding the key areas of sustainable communities, code modularity, project management, and test process management. To achieve high-quality software, OSS practitioners must fully understand these areas and how they relate to each other.

The sustainable community

High-quality OSS relies heavily on having a large, sustainable community to develop code rapidly, debug code effectively, and build new features. Many studies concluded that creating a sustainable community should be an OSS project's key objective.[1–3] An investigation of OSS project evolution found that a large base of voluntary contributing members was one of the most important success factors.[4] In particular, the study found that the system and the community must coevolve for the software system to have sustainable development and achieve high quality.

The onion model. In this model (see figure 1), the sustainable community consists of a small number of core developers and increasing numbers of contributing developers, bug reporters, and users.[5] "Onion" refers to the successive layers of member types. This is the most common model of a sustainable community.

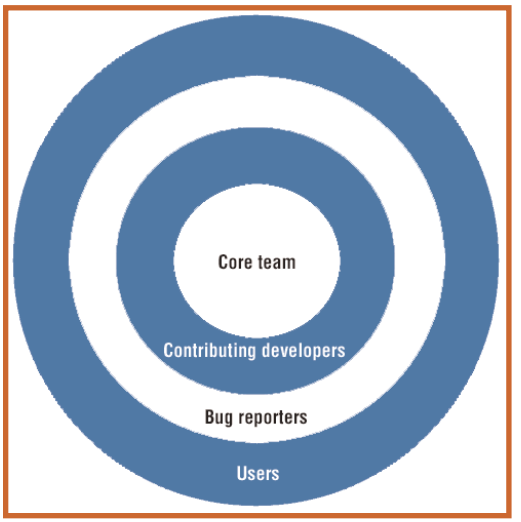

Figure 1. The onion model of a sustainable software development community.

Individuals create the sustainable community by increasing their involvement through a process of role meritocracy. As they move toward the core, users become bug reporters and might over time become contributing developers. A few contributing developers will eventually join the small team of core developers. Each type of member has certain responsibilities in the system's evolution, which relate to the system's overall quality. Advancement through the member types is reward and recognition for each member's abilities and achievements.

The onion model has three primary characteristics:

A sustainable community relies on attracting volunteer programmers.

- *The core team must be small*. As well as performing the bulk of the coding activity, the core team will exert control over the core system to maintain high modularity. They integrate only high-quality code, determine and follow the project roadmap, and maintain a fast, iterative release cycle. In a case study of Apache, a 15-person core development team performed over 80 percent of the functionality coding.[6] Most core teams will be much smaller but will perform a similar amount of work.
- *Contributing developers add and maintain features*. The ability to easily add new features depends on code modularity. Contributing developers tend to choose areas that are outlined in the project roadmap or that "scratch an itch." They'll also undertake bug fixes, peer-review code, and help ensure that the core team isn't overwhelmed with bug fixes.
- *Bug reporters take ownership of system testing and bug reporting*. The core or contributing developers will do few of these tasks. The sheer size of the bug-reporting group will ensure that more people test the system (on more platforms) than any commercial organization could hope to achieve. This group plays a key role in reducing defect density.

Participation and motivation. A sustainable community relies on attracting volunteer programmers. Much research has been done into what motivates the volunteer community; OSS managers must have a strong awareness of this area.

While high code modularity reduces complexity and lets newcomers contribute peripherally without impacting the core system, modularity alone won't attract programmers to a project. In Audris Mockus and his colleagues' case study of Mozilla—a highly modular project that initially had trouble attracting contributors—participation increased only after the core team improved documentation, wrote tutorials, and refined development tools and processes.[7] Other key motivators included opportunities to learn new technologies and tools and to take on new challenges.[8]

OSS teams must understand that programmers rarely volunteer their services selflessly—they wish to gain status in the community through reward and recognition. Various studies have concluded that

- programmers are looking for social status, which is "determined not by what you control, but by what you give away," and so strive to develop high-quality code;[2]
- recognition and flattery by a community giving credit where it thinks it's due—achieved by publicly naming contributors and letting people sign their own work—helps keep contributors motivated;[8] and
- as a project snowballs and becomes more successful, it will attract more developers who wish to bask in the project's glory and get some of the attention.[9]

A further motivation that isn't mentioned in the reviewed research papers and that could be an area for further investigation on this topic is that contributing developers often turn peer reviewer for subsequent contributions that make use of their code. This is important where the new code might affect their feature or other related functionality and they might wish to avoid blame for any problems that occur.

Code modularity

Gwendolyn Lee and Robert Cole's study of the Linux kernel development further supported the premise that code modularity lets many programmers extend the program by working on separate modules, without needing to change or understand the core system, or interfere with each other's progress.[10] This reduces the risk of new bugs being introduced in other modules.

The study of the Apache and Mozilla projects also concluded that both projects' low defect densities resulted from employing high code modularity and many bug finders and fixers.[7]

The largest study to date on code modularity's impact on OSS quality investigated 100 open source C applications and established a clear relationship between high modularity and quality.[1] The study described how small component size derived from good design and resulted in low defect density and high user satisfaction and facilitated maintenance and evolution. It would be worthwhile for further research to focus on open source applications developed in other programming languages.

Another study of the Linux kernel development concluded that modularity let multiple developers work on the same solution, often in competition, increasing the probability of timely, high-quality solutions.[11] Others have pointed out that bug fixing can also become a

competition to come up with the best, most efficient, and longest-term fix, so a high-quality fix tends to be the outcome.[12]

Project management

Two areas in which OSS development differs significantly from closed-source development, and which impact the level of quality achieved, are peer review and people management.

Peer review. OSS peer review assesses whether a contribution merits acceptance into the codebase.

Eric Raymond asserted that the rapid release cycle facilitates peer review because implementing and responding quickly to peer reviewers' comments and code keeps them involved and interested.[2] This results in a product that grows and extends rapidly and reaches high quality quickly. Conversely, Steve McConnell noted that while large numbers of peer reviewers are clearly fast and effective, they aren't necessarily efficient and that best practice software engineering indicates five to six peer reviewers as optimal.[13]

Raymond coined "given enough eyeballs, all bugs are shallow,"[2] meaning that if enough people see a software error, at least one of them will probably understand the error's causes and be able to fix it. This is particularly important given that a Linux kernel study concluded that 75 percent of the development work is typically mundane, labor-intensive tasks such as debugging, code reviews, and fixes. It's here, where OSS projects have access to far more human resources than closed-source projects, that OSS development demonstrates its power to develop faster than the closed-source model and to increase quality.[10] Cristina Gacek and Tony Lawrie observed that OSS development faces fewer time and cost pressures than closed-source development, so while a larger number of reviews might not necessarily be efficient, this might not be an important issue.[14]

Of course, commercial development has used code inspection for decades. A study of how to harness OSS development methods at Hewlett-Packard noted that code inspection became a continuous process in OSS development, whereas it tends to be a one- or two-step process at best in traditional software development.[15] Others conclude that peer review by people outside the core project team, without a vested interest in turning a blind eye, contributes greatly toward higher software quality.[16]

In an extensive study of the Apache project, the core team used the developers' mailing list to invite peer reviews, which resulted in many members outside the core community giving useful feedback on changes before formal release.[6] This shows how an OSS project can fix defects early in the life cycle, which McConnell had declared a major inefficiency in OSS development.[13] The rapid release cycle from an early prototype onward does support finding defects early in the project life cycle, although these might be found even earlier with more robust documentation.

Overall, the research agreed that peer review by numerous reviewers, particularly when carried out in tandem with a rapid release cycle, can significantly, positively impact software quality.

People management. This aspect of project management plays a vital role in developing high-quality software. This includes establishing an effective environment and culture, which some believe to be as important as system design.[4] Gacek and Lawrie stated that OSS development actively encourages innovation and creativity, creating a more people-focused process than traditional development, which is dominated by methods, tools, and techniques.[14]

In OSS development, the people-focused approach means that people volunteer to join a team and offer to work on a specific functionality.[4] Yutaka Yamauchi and his colleagues argued that this results in a culture of spontaneous work that is coordinated afterward—a major shift from traditional development where coordination and planning come first.[17] They summarized the OSS process as action, experimentation, and innovation followed by coordination, peer review, improvement, and stability.

However, other studies have found coordination occurring before action. In the Apache case study, when the core team found alternative solutions to a problem, they invited feedback from the developers' mailing list, and the core team then decided on a solution before it was developed.[6]

Clearly, you can manage work in various ways, and many studies have found clear, rational decision-making occurring in OSS projects, with sophisticated process-tool support to enhance collaborative development and debugging. Tim O'Reilly observed that a community needs to develop processes for voting on new features to decide who has access to the source

A people-focused approach means that people volunteer to join a team and offer to work on a specific functionality.

tree and to communicate in ways that don't stifle free-floating development.[3]

Testing process

In closed-source development, the testing process's rigorousness is arguably the most important factor in achieving high-quality software. In open source development, the success of error detection and fixing throughout the life cycle, and in particular during system testing, depend much less on procedural rigor. Instead, it employs a variety of techniques to achieve high quality.

Error detection and fixing. Obviously, OSS isn't fault-free. (For more information on determining whether OSS is of high quality, see the related sidebar.)

In a study of 200 OSS projects, Luyin Zhao and Sebastian Elbaum found that instead of focusing on high-quality milestone releases, the "release early, release often" process results in continual improvement by a large number of developers contributing iterations, enhancements, and corrections.[18]

Raymond's "given enough eyeballs, all bugs are shallow" relies heavily on numerous contributing developers, without whom, Mockus and his colleagues argued, the core developer team will become overburdened with testing and fixing, and code might never reach an acceptable quality level.[6] This reinforces the sustainable community model's importance in achieving high-quality OSS.

System testing. The study of 200 OSS projects discovered that

- fewer than 20 percent of OSS developers use test plans;
- only 40 percent of projects use testing tools, although this increases when testing tool support is widely available for a language, such as Java; and
- less than 50 percent of OSS systems use code coverage concepts or tools.[18]

OSS development clearly doesn't follow structured testing methods, but it can develop high-quality software. As with project management, software testing occurs; it's just done differently.

According to one study, the user base performs the bulk of the system testing, sometimes even exclusively, as with Apache.[6] A large pool of bug reporters is therefore crucial, ensuring software testing on many platforms. Others have concluded that OSS generally has lower defect density than traditional software because the latter can't test as extensively.[7]

However, not all projects rely on user testing. A study of the Mozilla project by Christian Reis and Renata de Mattos Fortes showed that it had an unusually strict quality assurance process and dedicated test teams.[19] In this case, automated regression testing was crucial, with nightly regression test suites run for Mac, Windows, and Linux versions to ensure the current version's robustness.

The methodology an OSS project adopts will depend largely on the available expertise, resources, and sponsorship. Formal testing techniques and test automation are expensive and require sponsorship. Some high-profile open

Sourcing High-Quality Open Source Software

If you're evaluating software, you'll want to know that the programs that reach your short list are of high quality, and you'll want to know this before you actually try them out for your organization. So, how can you tell whether open source software is actually high quality? Several evaluation models can help you.

CapGemini's *Open Source Maturity Model* (www.seriouslyopen.org) grades software against a set of product indicators, including, but not limited to, product development, developer and user community, product stability, maintenance, and training. This commercial model includes the use of CapGemini's own OSMM experts.

Navica's *Open Source Maturity Model* (www.navicasoft.com/pages/osmm.htm) is a formal process, published under an open license, that assesses the maturity of the key elements, including software, support, documentation, training, product integration, and professional services. It then applies product element weightings and calculates the overall product OSMM score.

The Business Readiness Rating (www.openbrr.com) is a proposed evaluation model under development by a group of major organizations that seeks to expand on both previous OSMM models to develop an open standard for scientific OSS modeling and rating.

If you don't want to evaluate software yourself, you can also procure OSS from companies that offer warranties and support for enterprise-level OSS. For example, SpikeSource and Red Hat both offer OSS *stacks*—bundles of integrated software—that are certified to work together and be reliable for enterprise use, along with software warranty, full support, consultancy, and training services.

source projects can achieve this, but most don't, so the user base is often the only choice. However, if resources and sponsorship allow, a strong case exists for mixing all these techniques: structured manual testing, regression test automation, and informal user testing will all identify different types of errors.

To date, no formal studies have compared OSS quality costs and testing rigor in sponsored versus volunteer-driven OSS projects, likewise with open source versus commercial products. So, it's unclear which approach leads to higher quality. However, Barton Miller and his colleagues suggested that products developed and tested purely by volunteers might be higher quality than commercially developed products, but more work must be done to clearly link them.[20]

There's some debate over how quickly OSS testing captures bugs. Decades of software engineering best practices have shown that the earlier bugs are found, the cheaper they are to fix. McConnell argued that in OSS development, bugs get fixed at code level rather than at the design level, which flies in the face of best practice.[13] He asserted that insufficient upstream work (for example, system design) can sink a large project the size of Windows NT (noting that the largest OSS project, Linux, is a fraction of this size). However, others have maintained that code-level defects aren't captured late because OSS peer review and system testing occur from prototype onward in an unusually fast, iterative process, which avoids showstopper bugs appearing in mature, shipped products.[12]

Research limitations

The published research reviewed here has some limitations. Much of it deals with team dynamics and analyzing OSS development as a social phenomenon. This work is important, but it doesn't necessarily help us understand why successful OSS projects attain high quality. Although some landmark OSS quality studies exist, they're few, and much more remains to be done. I've identified topics that will help fill the gaps in the current body of research:

- software quality in similar products developed using open source versus those developed commercially—for example, customer relationship management or workplace productivity solutions;
- software quality in open source projects with differently sized, sustainable communities;
- software quality in sponsored, professionally resourced open source projects versus those that rely on volunteers;
- the effectiveness, and ideal mix of, the three major types of testing in OSS projects—structured testing by the project team, regression test automation, and user testing—in finding errors;
- the effect on the sustainable community model of a commercial organization joining the development effort;
- the importance of deadline-related procedures—such as risk assessment, goals and milestones, early defect discovery, metrics, and scheduling—to software projects that are free of time pressures; and
- extending the study of 100 open source C applications[1] to other programming languages.

Although some landmark OSS quality studies exist, they're few, and much more remains to be done.

This review of the existing body of research raised many questions but also found important guidelines:

- High-quality OSS relies on having a large, sustainable community; this results in rapid code development, effective debugging, and new features.
- Code modularity, good documentation, tutorials, development tools, and a reward-and-recognition culture facilitate the creation of a sustainable community.
- The system and the community must coevolve to achieve sustainable development and high-quality software.
- High modularity and many bug finders and fixers result in low defect density.
- Rapid release cycles keep code reviewers and developers interested and motivated, quickly resulting in new features and high quality.
- Code review by people outside the project team leads to independent, objective reviewing.
- The project team's environment and culture are as important as system design when creating high-quality software. This can be handled in several ways, but success depends on a highly organized approach, with sophisticated tool support for collaboration, debugging, and code submission.

About the Author

Mark Aberdour founded and runs opensourcetesting.org, a resource on open source testing tools for software quality professionals. By day, he is a technical producer at Epic, a UK-based e-learning and knowledge solutions company. His technical interests include software quality, educational technologies, and open source software development and tools. He received his master's in software engineering from the University of Brighton. Contact him at 183 Mackie Ave., Patcham, Brighton, UK, BN1 8SE; mark@opensourcetesting.org.

- You can rely on the user base for system testing, but given enough resources and sponsorship, you should complement this with formal testing techniques and regression test automation.

Further objective study to expand our understanding of OSS quality would not only help us to understand the phenomenon of open source but would also benefit practitioners in both open source and closed-source software communities and advance the business case for OSS adoption.

References

1. L. Angelis et al., "Code Quality Analysis in Open Source Software Development," *Information Systems J.*, vol. 12, no. 1, 2002, pp. 43–60.
2. E. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly, 2001.
3. T. O'Reilly, "Lessons from Open-Source Software Development," *Comm. ACM*, vol. 42, no. 4, 1999, pp. 32–37.
4. K. Nakakoji et al., "Evolution Patterns of Open-Source Software Systems and Communities," *Proc. Int'l Workshop Principles of Software Evolution*, ACM Press, 2002, pp. 76–85.
5. H. Annabi et al., "Effective Work Practices for Software Engineering: Free/Libre Open Source Software Development," *Proc. 2004 ACM Workshop on Interdisciplinary Software Eng. Research*, ACM Press, 2004, pp. 18–26.
6. A. Mockus, R. Fielding, and J. Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," *Proc. 22nd Int'l Conf. Software Eng.* (ICSE 00), IEEE CS Press, 2000, pp. 263–272.
7. A. Mockus, R. Fielding, and J. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Trans. Software Eng. and Methodology* (TOSEM), vol. 11, no. 3, 2002, pp. 309–346.
8. K. Kishida and Y. Ye, "Toward an Understanding of the Motivation of Open Source Software Developers," *Proc. 25th Int'l Conf. Software Eng.* (ICSE 03), IEEE CS Press, 2003, pp. 419–429.
9. M. Bergquist and J. Ljungberg, "The Power of Gifts: Organizing Social Relationships in Open Source Communities," *Information Systems J.*, vol. 11, no. 4, 2001, pp. 305–320.
10. G. Lee and R. Cole, "The Linux Kernel Development as a Model of Open Source Knowledge Creation," Haas School of Business, Univ. of California, Berkeley, 2000.
11. J.Y. Moon et al., "Essence of Distributed Work: The Case of the Linux Kernel," *First Monday*, vol. 5, no. 11, www.firstmonday.org/issues/issue5_11/moon/index.html.
12. T. Bollinger et al., "Open Source Methods: Peering through the Clutter," *IEEE Software*, vol. 16, no. 4, 1999, pp. 8–11.
13. S. McConnell, "Open Source Methodology—Ready for Prime Time?" *IEEE Software*, vol. 16, no. 4, 1999, pp. 6–8.
14. C. Gacek and T. Lawrie, "Issues of Dependability in Open Source Software Development," *ACM SIGSOFT Software Eng. Notes*, vol. 27, no. 3, 2002, pp. 34–37.
15. J. Dinkelacker et al., "Progressive Open Source," *Proc. 24th Int'l Conf. Software Eng.* (ICSE 02), ACM Press, 2002, pp. 177–184.
16. A. Fuggetta, "Controversy Corner: Open Source Software—An Evaluation," *J. Systems and Software*, vol. 66, no. 1, 2003, pp. 77–90.
17. Y. Yamauchi et al., "Collaboration with Lean Media: How Open-Source Software Succeeds," *Proc. 2000 ACM Conf. Computer Supported Cooperative Work*, ACM Press, 2000, pp. 329–338.
18. L. Zhao and S. Elbaum, "A Survey on Quality Related Activities in Open Source," *Software Eng. Notes*, vol. 25, no. 3, 2000, pp. 54–57.
19. C. Reis and R. de Mattos Fortes, "An Overview of the Software Engineering Process and Tools in the Mozilla Project," *Proc. Open Source Software Development Workshop*, C. Gacek and B. Arief, eds., Dependability Interdisciplinary Research Collaboration, 2002, pp. 155–175; www.dirc.org.uk/events/ossdw/OSSDW-Proceedings-Final.pdf.
20. B. Miller et al., *Fuzz Revisited: A Re-Examination of the Reliability of UNIX Utilities and Services*, tech. report, Computer Sciences Dept., Univ. of Wisconsin, 2000.